|  |
| --- |
| Ramapo college Of New Jersey |
| PC Overlay with Chat-room |
| CMPS 450: Senior Project |
|  |
| **Amanda Steidl** |
| **Spring 2017** |

|  |
| --- |
| The following documents describe the process and details of the PC Overlay containing a chat room. Each document will be dated, beginning with the start of the project to the end. |

# Table Of contents

|  |  |
| --- | --- |
| Content | Page Number |
| *Initial Specifications* | 3 |
| *Final Specifications* | 4 |
| *User Manual* | 5 |
| *File Layout* | 6 |
| *Completion List* | 7 – 9 |
| *Learning Process* | 10 – 14 |
| *Program Flow* | 15 - 19 |
| *Class Descriptions* | 20 – 21 |
| *Data Structures* | 22 |
| *Issues & Concerns* | 23 – 25 |
| *Testing* | 26 - 28 |
| *Final Summary* | 29 |
| *Work Log* | 30 - 33 |
| *References* | 34 |
| *Code* | 35 - End |

# Initial Specifications

February 6, 2017

# Visual Design Details

* Dark-theme automatically, because it’s easier on the eyes. (Colors will come later).
* Overlay:
  + Borderless – no bar at the top or surrounding “box” structure.
  + Opaque – dark grey color with transparent properties.
* Chat room:
  + Sharp square.
  + Decently sized text & buttons (i.e. the send button).

# Issues & Concerns

* Constantly listening for a key bind as a background process, how much will it bog down the computer?
* Key bind conflicts, if the user picks a common key bind, the overlay will appear each time it is used with unknown results to the original key bind’s purpose.
* Managing multiple connections on the server.
* Coding language choices can make the designing of windows difficult.
* Keeping the chat room window attached to the overlay, there may be a way to detach them upon application running.

# Base Details

* Coding Language: *C / C#*
  + \*need to look into the language choice more.
* Overlay made as a new window would be using *DirectX*
* Overlay consists of *2 windows*: chat room, and overlay itself.
* These windows will be “attached” to one another with the overlay as the parent window and the chat room as the child. The child should only appear when the overlay is available.
* Process will consistently listen for the key bind to be pressed.
  + Key-bind will enable the overlay and then disable it by hiding it in the background.
* Chat room will have a network server connection so the user can successfully send messages.
* Chat room window should have a hotkey for when the user presses enter to automatically send the message. (Link to send button).

# Stretch Goals

* Push notifications to the user.
* Light-theme and dark-theme options.
* Mute sound notifications
* Remembering chat history options
* Text options for the chat room window
* [Idea] Having Gmail attached to the overlay

# General Description

The PC overlay is an opaque screen that will be activated upon a key-bound press to appear on top of all of the open or active windows on the screen. When the overlay is active, the chat room will be visible to the user while there is no access to the background applications. The overlay will have basic settings to change the key binds to enable and disable the overlay. The chat room will require a network connection and a server to have conversation history.

# Final Specifications

# Learning Process

* Learning of new technologies such as DirectX or OpenGL to create windows on the current system.
* Depending on the coding language that would be easiest for the visual aspects of the project, it may be easier to create the project in one language instead of another.
* Finding ways to thread the process to avoid bogging down the machine.
* [Possibly] Communicating with the OS to listen for key binds.

May 2017

# Base Details

* Coding C# in the .NET environment utilizing WCF services and WPF forms.
* All screens on the overlay will be comprised of windows, which will be designed using XAML.
* The messaging system will consist of global messaging and direct messaging to other users.
* The way direct messaging will be implemented is based upon if there will be a friend’s list or just a global chat. The global chat will allow users to click on other’s to directly message them. If there is a friend’s list the direct messaging will be either through the global chat or through their own direct friends list.
* Since the server will not be outsourced, the connections will be made only on a local machine to other instances of the program running for debugging purposes.
* The semi-transparent screen will take over the screen of the user’s screen while activated. A Hot-Key will be provided to toggle the application on and off anywhere on the computer.

# Stretch Goals

* Push notifications to the user.
* Password and data encryption.
* Database holding usernames and friend lists.
* Detailed server information stored.
* Tutorial on each button on first load.

# General Description

The PC overlay is an opaque screen that will be activated upon a key-bound press to appear on top of all the open or active windows on the screen. When the overlay is active, the chat room will be visible to the user while there is no access to the background applications. The overlay will have basic settings to change the key binds to enable and disable the overlay. The chat room will require a network connection and a server to have conversation history.

# Visual Design Details

* Individual windows will have a black and white theme for convenience.
* Chat windows will have a list of people within the chat.
* Log in screen will be simple with just a username.
* Information options will be attached to the overlay which will be a semi-transparent screen.
* All designs of buttons will be done personally on Google Drawings.

# Issues & Concerns

1. [*Stretch*]Connecting the server with the database.
2. Misconnecting clients with each other.
3. Allowing users to disconnect from the global chat and keeping direct messages open.
4. The semi-transparent layer communicating with each single direct message that is open.
5. Keeping the hotkey from conflicting from pre-existing Window key combinations such as CTRL+ALT+DELETE, ALT+TAB, etc.
6. Having the buttons “anchored” to a set location on any size screen.

# Learning Process

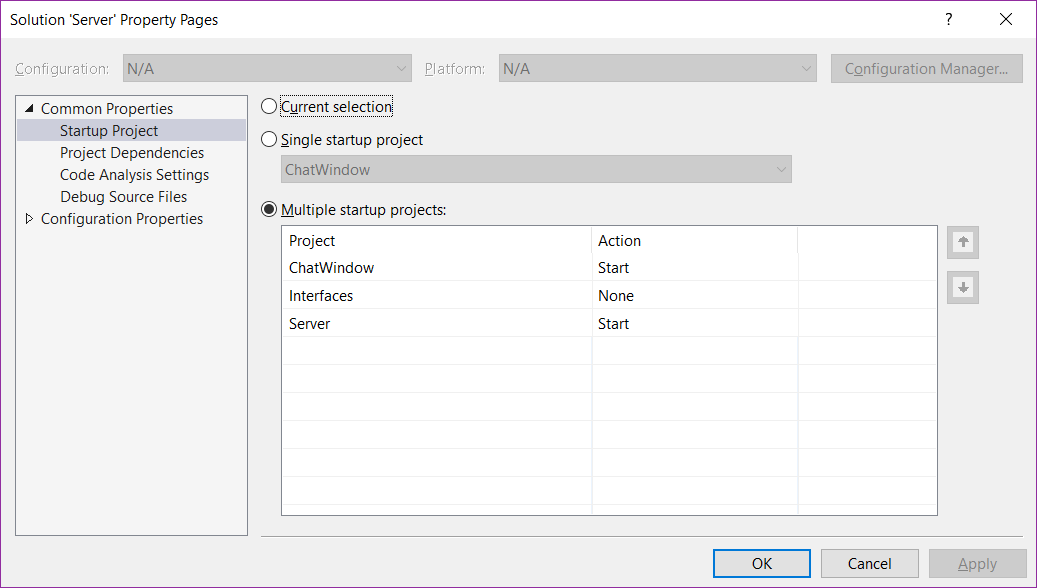
* How to properly use services integrating the interfaces and implementations properly for WPF and WCF.
* How to interact with the Windows commands and interrupt the messages for key combinations being pressed on any process and checking them.
* Manipulating WPF style windows and allowing them to communicate with one another.
* Creating a server for communications using C#, TCP, and maintaining multiple connections at once.

User Manual

Tools to have

* Windows Operating System
* Microsoft Visual Studio 2015, or newer

Running The Application through Visual Studio

1. Need to build the project with multiple startup projects
   1. Go to the solution, Right-Click
      1. Click on Properties [the bottom-most option]
   2. In the Properties window expand “Common Properties” and select “Startup Project”
      1. Select “Multiple startup projects:”
         1. Click on ChatWindow and Server.  
            Make sure the Action states “Start” next to them.  
            Interfaces should have the Action “None”
2. Once it looks like the above picture, press “OK” and now Debug/Run the application.

Hotkey While In Application

* To toggle the application while running press CTRL+Q

File Layout

* Each .xaml file has a corresponding .xaml.cs file within the same file path.

All contained within the same solution name Server

# Completion List

|  |  |  |
| --- | --- | --- |
| Task | Process | Completion Date |
| Server-side connection [Basics connection and messaging relaying] | Originally the server side connection was going to be using basic C# TCP connections. The other option was making a WCF application using C# in the .NET environment, and xml.  Testing the server was simple based on the frameworks in the WCF framework including Service Modules.  The modules provided give it an easy way to make a connection with an endpoint that a client or multiple can have access to.  The server code is only a few lines; however, the *services* it provides extensive functionality which will be expanded later.  Current stage:   * Message sending * Connecting everyone to each other. No direct messaging client to client.   **Alterations**   * Server receives requests to from a client to send a message to another client, will successfully send the message to the specified client. The destination client, if online and in the global chat, will receive messages; otherwise, sending messages would not be possible. | 05/13/2017 09/27/2017 |
| Client-side connection [Basics connection and message sending] | Client side involved creating a basic form application using WPF, a built-in option in Visual Studio in C#. This includes some code called xaml which is a mockup of xml and form attributes like in normal forms.  The client can log in, send messages and receive messages.  Once connected with the server it will get notifications of who connected.  **Alterations**  Client can now receive messages directly from other users in a separate window. Each individual user will create a new direct message window on the destination client side. | 05/13/2017 09/27/2017 |
| Physical Design of Windows | Login Window: The login window details will be in the service requirements section.  The login window contains a username input field and a currently disabled password input field. These fields will be passed back to the semi-transparent layer since it is the parent of all the windows. | 05/18/2017 |
| Welcome Window:  The welcome window’s behavior only consists of the username as a label welcome. [5/18]. Changes may come later to inform of missed messages while offline. | 05/17/2017 |
| Semi-Transparent Overlay Window:  This window is the parent/owner of all windows created at run-time. It will remain behind all windows created and be a semi-transparent solid color.  The layer has *all* information of the program and it is the one with access to the server since it communicates with all other windows.  **Alterations**  Removing Information Layer and adding the buttons to this layer. Reasoning: found a way to change the layer to semi-transparent while allowing the buttons to remain opaque. | 05/16/2017 09/27/2017 |
| Direct-Message Chat Window:  [removed]~~The window will be available for the user upon closing out of the welcome window or clicking the “+” Icon on the overlay “attachment.”~~  **Alterations**  The chat windows can only be opened from the global chat through right-clicking on the user desired and click “Message.” | 05/18/2017 09/26/2017 |
| [Removed]~~Information Window | Icons:~~  ~~This will be attached to the semitransparent overlay window, to provide an opaque interface for the user to utilize.~~  ~~Current buttons are the settings button and the new chat button.~~ | 05/18/2017 |
| [ Currently Unimplemented |  Code Exists |  Not tested]  Settings Window:  Allows user to disable or enable notifications. These changes will be applied and saved to files | 05/16/2017 |
| Icons & Design – Drawing & Implementation | Resource files addition to the client side chat windows and overlay interfaces.This enables the addition of personal images and files such as settings and username local saving. This is just for displays, to make the user interface appealing. | 05/15/2017 (ongoing) |
| Service Requirements | [Unimplemented]Design and options for editing options the settings of the user interface. File access for the file. | 05/15/2017 |
| Login capabilities [basic with just username]. | 05/15/2017 |
| Key binding for UI toggling. Key bind that was chosen for testing for clashing with Window key combinations /  CTRL+Q | 09/27/2017 |
| Global Chatting: Communication with everyone connected to the server. | 05/13/2017 |
| Direct Messaging: Communication with a selected person from the Global chat. | 09/27/2017 |

## Notes

* Section 1 and 2 should be completed concurrently to one another.
  + A client is needed to test the reliability of a server.
  + A server is needed to attempt connections with the client.
* Section 3 will be an overtime process since each portion comes with another part being ongoing or completed.
  + Multiple subsections may or may not be implemented, but will be crossed out upon being unimplemented.

# Learning Process

February 6, 2017 – Current

## OpenGL or DirectX?

* DirectX, even though OpenGL is slowly being supported across more platforms and hardware, DirectX has detailed documentation that is easier to understand in the time frame for this project.

## C# or C++?

* Originally attempted to code in C++ due to familiarity in the language, however, it proved to pose issues when trying to make the form to the specifications.
* In the .NET environment, specifically in Visual Studio, there is an easier, simpler way of creating and manipulating forms.

## Learning to Code Window Forms In C# February 20, 2017 - current

1. The GUI interface
   1. Dynamically adding properties to the window, as well as changing the sizing of them based on a screen size. Stack overflow assisted with this process.
2. The Window
   1. Created a window that maximized upon load and had an opaque appearance so the user can still see the windows behind.
3. Label onClick Methods
   1. Looking for a basic way to allow the user to change settings of the window (color, text size, etc.)

Design Implementation for Time

1. [Tool Option] Label to update on time.
   1. In the case of full screen applications, this could display the current time / date for the user based on the current system settings.
   2. To look up :
      1. How to get the system time
      2. Dynamically updating text each second.
      3. Formatting? HH:MM:SS
      4. Right now, I know how to move controls based off of screen; however, find a formula to keep at middle center [upon each "paint update" from the time]?
   3. Setting System Time
      1. Using Timer object with 1000ms tick (1 second) and setting the label within the timer with:

lblTime.Text = DateTime.Now.ToString("h:mm:ss tt")

* + 1. Make sure to start the timer on the form load or nothing will load!
  1. Setting the Positioning
     1. Grid Layout (xaml) [4]
        1. Each grid itself can be split into smaller grids.  
           General layout:

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width = ”int|double|\*”></ColumnDefinition>

<!--The \* following the numerical value will be a percentage of the screen percentage to take up based off of all the values in the height or width-->

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinitions Height = ”int|double|\*” ></RowDefinitions>

<RowDefinitions></RowDefinitions>

</Grid.RowDefinitions>

<!--This would be where to declare all the items to go within the grid layout-->

<TextBlock Text = “this one” Grid.Row = “0” Grid.Column=”0”/>

<!--The text block would be within the top right box [0][0] like a 2D array -->

<TextBlock Text = “spanning columns” Grid.Row = ”0” Grid.ColumnSpan = “3”/>

<!--The block will spawn the first rows three columns-->

<TextBlock Text = “spanning rows” Grid.Row = “1” Grid.Column = “0” Grid.RowSpan = “2”/>

<!--This would span 2 rows, you can also combine column span with row span, as well as specify the row/column to start -->  
</Grid>

Child Window For Settings & Chat-Room

1. Child Windows
   1. Creating a child window on the "onClick" method of a button or label.
      1. Required for: Settings options, & chat-room toggling.

[Summer Work Start]

Server-Side: Creating A TCP Listening Server

1. TCP Listener: starting a port on my personal machine. [2]
   1. **Port number – 3000**
2. C# Try catch block.
   1. Unable to reach variable declared within. Found that is must be declared as NULL before use. Then re-checked afterwards just in case, exiting the program accordingly. [3]

Using C# **WCF** To Create A Service

1. Server
   1. The server itself is created by a ServiceHost object from the WCF Service Model libraries.
   2. The Service model provides the frameworks for having endpoints which clients can connect to.
      1. These endpoints will be for the home machine so using localhost and the IP address 127.0.0.1 would be satisfactory.
   3. The server will remain open just using a line read.
   4. Background operation occur within the implementation of the **services**.
2. Service(s)
   1. Under the server solution there are two different implementations.
      1. One which manages connected clients
      2. Second that has implementations of all other “services” a chat client should offer to its clients/users. This includes, but not limited to upon further implementations:
         1. Sending global messages and receiving global messages.
         2. Signing in using a unique username based on global available usernames.
3. Implementation verse Interface - definitions and explanations.
   1. Terminology using WCF in C# is confusing and less straightforward than preferred.
   2. **Implementation**: *like a class cpp file where it implements the functions within the header files.*
   3. **Interface**: *similar to a header file where it just describes/lays out what is available for the user to access.*
   4. Unlike C++, the implementation and the interface are linked by adding:
      1. References: *right-click the project then add the appropriate reference to link the files.*
      2. Headers (“using”): *adding the appropriate using header to the same file*.
4. Marking Services within **Interfaces**
   1. When having a class that is an interface for the client/user it has to be marked as:  
      [ServiceContract]  
      public interface IntServices

Int is my abbreviation for interface on a class.

* 1. When having functions within that class it must be labeled as:

[OperationContract]

Datatype FunctionName(datatype params);

1. Implementing the Services defined in the Interface

[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, InstanceContextMode = InstanceContextMode.Single)]

public class ImpServices : IntServices

* 1. Above describes a service that will support multi-threading within the WCF framework.
  2. Implementation is abbreviated by Imp
  3. As previously stated the functions which will be called from the client are described within this class.

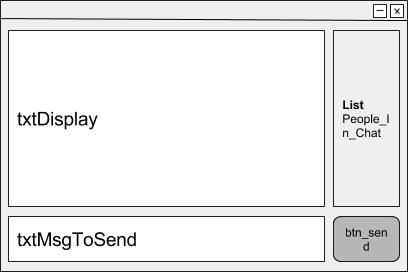
1. Storing General Application Information
   1. This involves configuring the specific files namely, App.config, within the server and the chat application window.
      1. This will hold binding information, not yet implemented.
      2. On the client end, it needs an endpoint address of the server.
      3. On the server end, it has an endpoint address, which for now will be the localhost.
   2. Other than the information above the App.config, will hold the links between the Interfaces and the corresponding Implementations.
      1. The current project has a link between:
         1. Server.ImpServices and Interfaces.IntServices
         2. ChatWindow.Implement and Interfaces.IntClient

**WPF**: Communicating Between Windows

* WPF is the Windows Presentation Foundation.

Using XAML and C# to Create **Events**

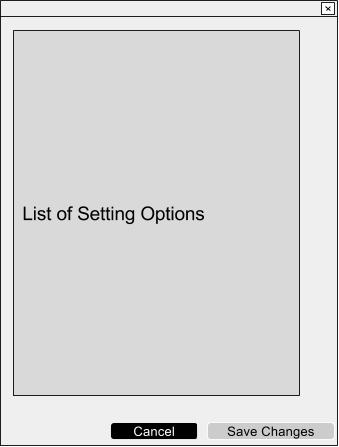
Beginning Design Choices: Modifying & Refining Current Functionalities

1. Splitting up functionalities of the window in designing. [Images not to scale]
   1. Chat Windows
   2. Separate Log in

**Design Reference 2|3**

1. At this point: Required settings functionality with a save file.
2. Button functionalities between server and client chat window(s).

**Design Reference 1**



**Design Reference 4**

Program Flow

Design Layout

The design of the application was from the backend to the frontend processes. Initially creating the server for the user, and laying out the functionality that would need to be accessible for each client. After the server was designed, it needed to have the functionality filled in; this had to be done after a basic client was made so the code for sending messages and connecting the client to the server. The windows for the client messaging had been created and simpler username input boxes. Once the user gained connection to the server it was easier to connect multiple clients at once and then messaging testing was available. After maintaining the server, the client-side design and spreading out the functionality was available to be done. Things like splitting the logging in windows apart from the messaging windows as well as creating the overall transparent layer. The communication between windows was done when each window was created and the only connection between windows that had to be maintained was the semi-transparent overlay and the other window. Once each window was connected correctly there was testing and small functionalities to be tested particular to each window.

**Server Layout**

**Window Data Layout**

Send to client if exists

Send to all except client requesting

Send Message To [client]

Send Message To All

Update Client-Side List [delete]ExistingClient

Remove

Add To

No

Already In Client Connections?

Yes

Login

Throw Error in Server Console, Assume Disconnect

No

Yes

Is the User Currently Connected?

Yes

Yes

Listen for Client Connections

Listen for Client Disconnections

Based off of the callback function used within the Client-Side program, it will directly request one of the services below.

Wait For Requests  
\*Note: Able to Handle Multiple at once

Yes

Update Client-Side List +newClient

<List> All Client Connections

Start-up Server

Each window passed its data or got the data from the semiTransOverlay window.

semiTransOverlay

* The overlay screen is the first to load on the client-side on the initial startup.
* The screen holds important information, and will pass information to all the screens that require details about connections or the user’s information.

< semiTransOverlay>

* Semi-transparent overlay is responsible for:
  + Login window creation/closure
  + Welcome window creation/closure
  + Global chat window creation/closure
  + Direct message window creation/closure

-->Login (window)

* When the semi-transparent overlay loads the login window will be the focused window to prompt the client for a username.
* If the username is unique, the client will successfully login or be prompted to enter a different username.

< semiTransOverlay>

* Upon the user successfully logging in from the server [see Server Layout] the username is passed to the semi-transparent overlay.
* From here the username can be passed to any window.

-->Welcome (window)

* The username is obtained from the semi-transparent overlay when the client is connected, and then the welcome screen just uses it to welcome the username.
  + This window in future editions may show updates/new features etc.
* Upon window close it will start an event in the semi-transparent overlay.

< semiTransOverlay>

* When receiving the message from the Welcome window closing, the semi-transparent overlay will start a new instance of the GlobalChat window.
  + The overlay will pass the: username, and connectedclients list when the global chat window.

-->GlobalChat (window)

* The username is obtained from the semi-transparent overlay like all other windows.
* The connectedclients are obtained from the semi-transparent overlay and is used to populate the list box on the global chat window. With each connection and/or disconnection the listbox will be sent updates from the semiTransOverlay. This allows one object to maintain a “master key” of what each window should look like/contain.
  + Each client listed in the box has been enabled to receive messages independently from all users in the GlobalChat window.
    - Using this functionality will open a DirectMessage window (through calling functionality within the semi-transparent overlay) for the user to use to contact the selected person in the list.
* When sending messages, the GlobalChat window calls the parent’s (semiTransOverlay) public function to send the message to the list of users.
* When *receiving* messages from the server, the GlobalChat will be updated through the semiTransOverlay message forwarding functionality.

< semiTransOverlay : Sending Messages to all/Global Chat>

* When sending messages to another client the semi-transparent overlay will use the connection with the server in order to send a request to message **all** the users currently online.
* The server will receive this request and respond by going through the list of users and sending the message to each user.
* Upon receiving the message the semi-transparent overlay will get the message and display it through the GlobalChat window’s display textbox.

< semiTransOverlay : DirectMessage>

* When opening a new DirectMessage window the window(s) will be managed in a dictionary with the recipient’s username as the key.
  + This allows the messages to be received easily from the correct username. Sending the information from the overlay to the DirectMessage window makes it easier to maintain a larger list of direct messages as well as less confusion of how to manage them independently of one another.

--> DirectMessage (window)

* When receiving a message from another unique username, the semiTransOverlay opens a new DirectMessage window to accept an incoming message or update a current chat with the message.
  + This process could be endless if one were to receive messages from infinite numbers of users.
* The DirectMessage window behaves the same way as the GlobalChat window except it has one sender and one target rather than one sender to multiple targets.

< semiTransOverlay : Sending message to a single client/ Direct Message>

* When sending a message to another client, the semi-transparent overlay will use the connection with the server to send a request to message a **username** if online.
* The server will receive this request and if the user is available/connected will send the message to the user.
* Upon receiving the message the semi-transparent overlay will display the message to the DirectMessage window corresponding the username the message is from(the key).

Class Structure

# ServerMain

* The class the server is contained in.
* Smallest of all the classes; however, this class is where the initial instance of the implementation of services is[ImpServices()].

<Summary>

The server is started here using TCP style connections and the host machine as the server address. This class is for just starting the server and keeping it open through a simple ReadLine command. The bulk of the server-side code will be in the implementation of services, or rather the server responses to client requests.

</Summary>

Window-Based

## SemiTransOverlay

* The parent of all the windows that are created in this application.
* Directly containing:
  + Buttons for window options.
  + Button for disconnection.
* Indirectly has access to and creates:
  + Direct message windows. Maintains a window list for the client.
  + Global chat window
  + Log in window
  + Welcome window
* Initial window loaded:
  + Login
    - Welcome
      * Global Chat

<Summary>

The overlay itself is a semi-transparent black window that contains and creates each aspect of this application. The application listens for the application’s hotkey combination of CTRL+Q. This hotkey combination will allow the client to hide the entire application from view, while also focusing it into view upon toggling again. The child windows can be manipulated through this class in order for all the client-side services to be located in a single location. The other windows are only able to complete the basic tasks and send all information gained back to the parent window, such as text-input and send requests.

</Summary>

## GlobalChat

* Global chat is opened after the user successfully logs in then closes out of the welcome screen.
* Global chat contains:
  + A list of all users currently connected and using the global chat room on the server.
    - The list itself has a drop down menu with the ability to message a user directly. If multiple are clicked on it would just be the last one that had focus.
  + Ability for the client to type and send messages to those connected.
* Size of the window is fixed.

<Summary>

The user can send messages to all connected to the server while also having the option to send a message to a single user. The window can be opened, closed and minimized from the semi-transparent layer.

</Summary>

## DirectMessage

* Created upon the user requesting to send a message to an individual client on the global chat screen.
* When a message is sent, the client receiving the message will have a new window popped up on their screen.

<Summary>

Simply enables the user to communicate with another single user chosen from the global chat window.

</Summary>

## LogIn

* A window to log in the user with a unique username, if duplicate will continue to ask.

## Welcome

* Welcomes the client by username.

# Implementations:

## ImpServices

* Server-side services for:
  + Sending messages and receiving messages.
  + Getting client connection and disconnection updates.

<Summary>

The server is set up with a list of functionality such as receiving send requests from clients then sending them to the specified client(s). The server keeps track of each connected client and with their username information. The server is connects clients making sure there’s no duplicate usernames existing, as well as disconnecting the user and removing them from the user list. Upon connection/disconnection from the server, it will be forwarded to all the other users.

</Summary>

Data Structures

# Dictionary

* Dictionary<string, DirectMessage>  
  string for the user name of the contact whom the conversation is with.

DirectMessage is the corresponding direct chat window associated with the user name.

The dictionary was used for quick access for chat windows that are currently open to enable faster focusing and ordering when implementing a fast-access button for alphabetical chat window access.

## Concurrent Dictionary

* ConcurrentDictionary<string, ConnectedClient>  
  When clients connect to the server, they would be put onto this list. The concurrent dictionary is just the thread-safe version of a normal dictionary, preventing duplicate/collisions occurring during data input from users logging in. There are still checks for these mishaps; however, the concurrent dictionary will more than likely prevent the collisions from happening.

# Duplex Channel Factory

* DuplexChannelFactory<IntServices>  
  Channel factory which contains a reference the interfaces and implementation of services the client has access to as well as the endpoint for the client-side. The services that are implemented are accessible through this by creating a “channel” between the interfaces on the frontend and the implementation on the backend.

# Lists

* List<string>  
  strings of the connected users on the client-side, this would be passed by reference to the server for the initial client list that is required by the client.

# Observable Collection

* ObservableCollection<string>  
  In order to use a list efficiently in a WPF ListBox the normal list had to be transferred to an Observable Collection, which in itself is a list of objects or “item” like a list is. The observable collection can be set to a pre-existing list, then used as a ListBox’s items source where it will update as soon as the observable collection is then can be sorted.

# Issues & Concerns

February 14, 2017 – Current

1. May 2017: Issues in C++  
   When adding in :

HRESULT hr = DwmExtendFrameIntoClientArea(hWnd, &margins);

if (!SUCCEEDED(hr)) {

exit(0);

}

using the debugger for the solution had odd behavior. Meaning the solution had to be built **then** ran in order to have a *completely transparent window*.

Note: that without the below line of code, following the above, is required for a transparent window:

SetWindowLongPtr(hWnd, GWL\_STYLE, 0);

Without the line of code, the program will show the color specified for the hbrBackground color in the window properties at the start of WinMain.

1. May 2017: Changing from C++ to C#

Using C++ became a problem trying to separate each Window and trying to parse out code that was necessary in one window but not the other. The Window to Window communication became a difficulty that did not seem necessary knowing there are other tools such as C# in the .NET environment which had all of the tools and more necessary to complete the project.

1. [Solved] March 2017: Semi-Transparent Window Layers  
   When manipulating the code to try to enable the controls on the semi-transparent form to be completely opaque, there became issues of choosing one option over the other as opposed to a combination of the two.  
     
   To appropriately fix this problem to get the design correct, there will have to be two separate forms: one for the opaque background with a transparent window on top with all the controls.  
     
   Solution Moving from C++ to C# with XAMLUsing XAML there are different tags associated with the <Window> tag. The <Window> tag without any modifications to results in the whole window and its contents to be semi-transparent.

Using <Window.Background> with a nested <SolidColorBrush> tag it will just change the background of the *main* window. Normally, the <Window> is a container for the rest of the nested tags making the entirety of the page transparent. Example:

<Window.Background>

<SolidColorBrush Opacity="0.7" Color="Black"/>

</Window.Background>

1. September 2017: Hotkey Conflicts – Pre-existing hotkeys

Since my program was only checking against the Windows key combinations such as CTRL+ALT+DELETE and ALT+TAB, as well as Google Chrome key combinations such as CTRL+T, CTRL+W, CTRL+N the key combination used for the program is CTRL+Q. If this key combination is altered or changed within the code to a pre-existing key-combination on the system using the program there will be unhandled errors or exceptions thrown since the key combinations will conflict.

1. [Solved]September 2017: Client Disconnection

When the client disconnected from the server there was no initial record of the disconnection notification to the other clients; therefore, it would say that the individual is online when they were disconnected already.

Solution: Update the User List to all Connected Clients

Upon connecting or disconnecting the client disconnecting would be updated to all the clients before the user successfully disconnects or after they successfully connect to the server.

1. September 2017: Running 2 users on the same machine – Hotkeys  
   When debugging on a machine, the hotkey in use would bind at an operating system level, preventing other applications which start up from using that same hotkey. An error would be thrown and terminate upon a new instance of the chat window, or client-side, application being started.

There is no work around with this, the program had to be tested with and without the hotkey code installed.

1. September 2017: Leaving the Application through Windows OS Hotkey

An individual cannot click outside of the application’s window or focus the background Window, which was the purpose of the program. However, if an individual uses the Windows hotkey ALT+TAB they can get out of the application making it run in the background of other processes. The hotkey command for hiding the process will still work; however, this is not the intention of the program.  
Research done on this issue has proven that preventing ALT+TAB from being used in the application would prevent it from being used anywhere while the application is running. Therefore, nothing was changed.

1. September 2017: Debugging Application

Functionality of the program could not be tested all at once due to the hotkey conflicts when two applications ran at the same time [see Issues & Concerns #6]. Therefore, if there is conflicting functionality between parts of the project with receiving and sending messages between the client and server with the hotkey it has not been tested. Clients sending messages to the server works fine; therefore, it is assumed if deployed on separate machines the clients would be able to send messages and use the hotkey.

1. September 2017: Windows Operating Systems Limited

Application is only available to Windows Operating Systems since the DLL requires Windows style messages and hexadecimal key codes in order to function properly. These messages are for checking against registered hotkeys / unregistering (the applications) hotkey when it closes.

Testing

* Server connection
  + Client connection testing to see if a **single** client could make a connection to the server.
  + **Multiple** client connection by starting up multiple overlays and having the console print who connected/came online.
* Server interface/implementation
  + As clients connected maintaining a list of who has connected / timing to make sure there was no lag/interruption between connections.
    - Printed list to console. [This list was only printed during testing, no longer exists]
  + Client disconnection printed to console.
  + Printing messages to the console and whether it was directly to another connected client or to the global chat(all clients).
  + Logging into the application with identical login names because each username needs to be unique.
  + Sending proper notifications for client connection/disconnection to the users on the connected client lists.
    - Ensuring the proper amount of notifications were sent, if at all (if no one else was connected to the server)
  + Making sure the client was fully disconnected from the server. Checking if they were in the list after the disconnection occurred.
* Client-side logging in
  + Logging in with a duplicate username should not terminate application but prompt the user to enter a new/different username.
  + Checking the connection between the clients to the server through message “pings” i.e. treating the username input as a message sender to the server. [For testing purposes, not implemented this way in the final solution].
  + Logging in should open up global chat, this is treated through event handlers. When initially created having the Log-in screen close as an event to ensure the global chat box to open did not always open. The event seems more reliable.
* Client-side messaging (multiple and single client connection testing)
  + Taking messages from the client side (any form of message) and getting it to the server. Watching the server print messages out for a “success” to the server.
    - No message character limit probably should have one.
  + Client-to-All was having sending a single message sent to multiple clients. This was through having multiple instances of the chat window open and seeing if the users were successfully connecting through the server by the username/connection dictionary combination on the server-side. The message was either received by all, or if blank received by none.
    - Clients can not send blank messages of any form to the server. Tested through sending spaces / enter keys / tab characters to the server.
  + Client-to-client having the server find the correct key. If the key is no longer in the dictionary an error was thrown. Fixed by checking if the key existed then sending if it was in the dictionary.
    - Possibility if timed correctly an error can still occur between the user disconnecting and a message sending to the client disconnecting from the server.
* Client-side chat toggling
  + When toggling the chat there was an issue with the client “disconnecting” from the chat entirely and then going back to global chat with no user list. This issue was resolved by refreshing the current connected client list again on the client re-connecting to the server.
  + Another issue occurred and resolved was the user re-connecting and then their username would appear twice to other users. This was due to the initial disconnection occurring but the updated list [without the one client] was never updated to the rest of the connected clients.
* Client-side hotkey for toggling application
  + Hotkey conflicts with Windows operating system key binds. This was avoided by creating a predefined hotkey CTRL+Q which did not conflict with any Windows operating system running programs.
  + Hotkey creation had to be through Windows operating system message passing, and tested through binding the hotkey multiple times and testing to see when errors were thrown.
  + Attempting the hotkey combination while focused on different Windows while the application was not visible to the client:
    - Being on the Windows Desktop
    - Clicking on Visual Studio
    - File folder
    - Google Chrome
* Window Styling and Button/Feature Testing
  + Windows: Login
    - Making sure the login used the event within the semi-transparent overlay when attempting to login so the overlay could query the server for duplicate usernames or usernames that were currently incompatible with the server.
  + Windows: Direct Message
    - Message displaying
    - Username of chat buddy displaying within the title bar
  + Windows: Global Chat
    - Message displaying, button functionality and communication with the semi-transparent overlay. The function calls and buttons were also used for the direct messaging window.
    - List box communicating with its special list object of items for the usernames to appear.
    - Making sure that when using the list box and selecting the user, the correct user was pressed.
    - Making sure a direct message window would appear when clicking on the “message” option on the dropdown menu from each individual.
  + Windows: Welcome
    - Making sure the window appeared after the login window. Properly formatted username, etc.
  + Windows: Semi-Transparent Overlay
    - Display testing. Button locations, layout testing, etc.
    - Image vs. Button functionality testing, bitmap to show hovering color changes for buttons etc.
    - Connecting the messaging windows information with the overlay. Back and forth “communication” through functions was required since the server object was located within the overlay and not the independent chat windows.
* Unable to be tested
  + Users directly messaging another user while the client receiving the message has the application toggled “off.” Due to the hotkey conflicts while running multiple client instances on one machine, this was not handled in the code and therefore the behavior is unknown.
    - New chat may be opened in view of the entire screen while the semi-transparent window is hidden which would be incorrect.

Final Summary

Work Log

# February 6, 2017 – Current

## February 6, 2017

* Specifications are written, considering the coding languages as well as the technologies to use.
  + C-style language with either OpenGL or DirectX.

Week 01

## May 08, 2017

* After working with Windows Forms in the .NET C# environment, it is apparent that the language I should be using is C#.
* Today began the day of combining the bits and pieces of the project I've been doing in independent projects. Including but not limited to:
  + "Attaching" or relating the forms to one another.
    - Base form: semi-transparent
    - First child form: controls for manipulating the layout (implementation is currently a work in progress)
  + Opacity of just the base form.
  + Removal of the icon from the taskbar.
    - For whatever reason the removal of the icon from the taskbar prevented the forms from splitting into multiple independent forms.
  + Adding the current time to the child form.

## May 10, 2017

* Attempts to create a server using just the native C# .NET environment tools through TCP style sockets and connections.
* Managed to create a server and have a client connect; however, the asynchronous call was not working as planned.
  + This caused looking for other ways to create a server in C# using Visual Studio.

## May 11, 2017

* Learning what a WCF service is, and how it worked to create a connection between the server and client. Learning was provided through Youtube and the official MSDN website for C#.
* In the learning section, filled into what was found from WCF, Windows Communication Foundation, services and WPF style forms. The forms are like normal windows forms, it provides an interface to be used within VS Blend as well as in Visual Studio.
  + Provides multiple functionalities such as security and multithreading built in.
* Watching tutorials online on how to properly connect the Server with Services and the Services with Clients.
* Considering the use of a database, to store a finite number of usernames as well as their corresponding friends lists.

## May 12, 2017

* Designing the Server, along with the chat window for messaging purpose.
  + Connection through TCP connection using localhost
* Designing the chat window to have the functionality listed:
  + Login
  + Typing and sending message
* The services implemented on the in between, allowing for messages to be relayed and for the login process to ensure only one username is currently active on the server.
  + Created the service implantation and corresponding interface.
* Testing connection with the client and server.
  + The connection made; however, not yet tested on multiple connections through the server.

## May 13, 2017

* Listing out the functionalities that need to be modified on the chat window.
* Modifications will allow the interface to be cleaner and more user friendly.
  + Allowing for:
    - Inability to send messages unless logged in.
  + Ideas to implement:
    - Saving username internally on a folder.
      * This would probably be attached to settings.
    - Database with the list of usernames, friends list, etc.
    - Time stamps on messages or option for.

Week 02

## May 14, 2017

* Drawing the designs for the new windows such as:
  + Global Chat Window
  + Direct Message Chat Window
  + Log-in Window

## May 15, 2017

* Computer design of the drawings, within the design section of learning.
* Separation of current code:
  + Settings window (unimplemented)
  + Overlay – Semitransparent
  + Log in screen, password currently not implemented.
* The overlay can create the Log in screen on top of it.
  + Therefore, every other screen that needs to be created will be simplified.
* Creation of personalized buttons – clickable images and a simple Icon for login screen.
  + Might port these to buttons later
* Learning much about how the WPF window style forms are different than the normal C# forms within Visual Studio.
  + Use of MSDN, Stack Overflow, C# code project, Youtube, etc.
* Completion of code design of:
  + Log in screen.
  + Overlay (it is just semi-transparent window, with no border)
* Ability to log in with just a username: no way of checking uniqueness yet.

## May 16, 2017

* Documentation of what has been done. Marking areas of learning, need to know, etc.

## September 23, 2017

* Reading through the code to check what is left, refining code.
* Debugging code to make sure everything is running properly.

## September 24, 2017

* Server functionality such as:
  + message passing to all and to individual
  + updating connected list to all clients
  + disconnecting
* To debug the above functionality, have to enable client side callback functionalities

## September 26, 2017

* Moved buttons from the information layer to the semi-transparent layer.
  + Removed the information layer entirely.
* List of clients connected to the server on the global chat.
  + Debugging the list while clients leave/close the chat.
  + Fixing a bug where the user duplicates on re-entry into the chat.
* Adding a drop down menu to message list of users in the global chat
* Debugging direct messaging users, making sure the right user gets selected

## September 27, 2017

## [Continuing]Debugging multiple user connections as well as message sending

## Debugging messaging and formatting of messages

## Adding disconnection button for the user’s use

* Disconnecting the user through the server based on user name, checking if username is still in the connected list on disconnect, error checking.
* Registering/Unregistering hotkeys and using them within the project with multiple/single windows open.

## September 28, 2017

* Last day of debugging and testing the code before presentation.
  + Messaging individuals.
  + Messaging multiple people in separate direct message windows.
  + Messaging global chat that has 4+ clients connected.
  + Disconnection from server and what should happen if the user doesn’t properly disconnect.
* Presented completed and implemented portions of the project.

## September 30, 2017

* Documentation

## October 1, 2017

* Documentation

## October 2, 2017

* Documentation

## October 3, 2017

* Coding documentation through the built in C# html style tags.
* Moving code around to flow better through each section
  + Labeling sections so it is easier to find code that is grouped in one overlying topic.

# References

1. "How do you get the current time of day?", StackOverflow, http://stackoverflow.com/questions/296920/how-do-you-get-the-current-time-of-day
2. “TcpListener Constructor(IPAddress, Int32)”, MSDN, <https://msdn.microsoft.com/en-us/library/c6z86e63(v=vs.110).aspx>
3. “How to avoid 'Unassigned Local Variable' defined inside a try-catch block”, StackOverflow, http://stackoverflow.com/questions/16999410/how-to-avoid-unassigned-local-variable-defined-inside-a-try-catch-block.
4. “Grid Layout”, C# Corner, <http://www.c-sharpcorner.com/UploadFile/1e050f/grid-layout-in-wpf/>

# Design References

1. “Log In Screen,” Amanda Steidl. May 15, 2017.
2. “Global Chat Screen,” Amanda Steidl. May 15, 2017.
3. “Chat Window,” Amanda Steidl. May 15, 2017.
4. “Settings,” Amanda Steidl. May 15, 2017.